Amendments to the Specification

Please replace paragraph 79 starting on page 22, with the following rewritten paragraph:

[0079] The probe controller is also preferably an RMI service registered to kernel 2530, and starts and stops other probe components gets the probe's configuration. New configurations of probes are sent from kernel 2530 to the probe controller. The probe controller will determine if a probe should be started or stopped or a filter changed upon receiving a new configuration. A logic diagram for the Probe/Publish Engine is provided as Fig. 10. In accordance with this logic diagram, when the JVM is loaded, at shown at 2702, events identifying thread starts, thread end and completion of initialization of the JVM are registered through JVMPI to the kernel, as shown at 2704. When initialization of the JVM is completed, as indicated at 2706, then the system properties are set and a thread for initiating probes is created, as indicated at 2708. The thread then enables the method entry and class load events, as indicated at 2710, and waits until the application server has started, as shown at 2712. A method entry starts a process flow in the JVM, at 2714, which checks to see if the application server is started, at 2716. If it is started, the process flow is passed to the InitProbe Thread, which disables method entry and class load events, at 2718, and creates a probe controller thread, at 2720. The probe controller thread seeks a probe configuration from the kernel, at 2722. If the probe configuration is not found, then the flow ends, as indicated at 2724 and 2726. If the configuration is found, the process flow proceeds to determination if the probe is enabled in the configuration, at 2728. If not, then the process flow ends. A process flow may also commence with a new configuration in the form of an RMI call from the kernel, as indicated at 2730. If the probe is enabled, then the process flow proceeds to start the event agent and command agent, enable a class load events and a method entry events, as shown at 2732. The command agent awaits a command from the visualization engine, as indicated at 2734 and 2736. The enabling of the method entry event starts a process flow in the JVM, as indicated at 2750. Data, such as CPU clock, wall clock, method identification, thread identification, and/or URL and SQL are obtained, as indicated at [[2750]] 2752 , and passed to event queue 2760. The class load event initiates a process flow in the

JVM, as shown at 2754. A function of get class name, method name and signature is initiated, as shown at 2756, and this information is passed to class hash table 2762. The event agent retrieves records from the event queue 2760, as indicated at 2780. The event agent will wait depending on the publishing frequency, as indicated at 2782.

Please replace paragraph 80 starting on page 23, with the following rewritten paragraph:

[0080] Visualization engine 2750 provides the front end user interface component used in the method and system of the invention. Standard J2EE technologies may be used for implementation of visualization engine 2750. The front-end framework of visualization engine 2750 handles housekeeping such as session management and security. The visualization engine 2750 preferably handles as many common tasks as possible in order to provide an environment conducive to the development of front-end application and business logic components. The visualization engine 2750 sits on top of a database, which it accesses in response to user requests. The architecture is illustrated at Fig. [[28]] 11 , and is shown to be browser-based, using a browser 2810, communicating with a web server 2815, which may be an Apache web server, and an application server 2820, such as IBM's Websphere, interfacing between the database 2560 and the web server 2815. Servlets may be used to handle requests and manage application flow. Servlets may also be employed to control front-end behavior by performing form data-entry validation and sending java bean objects containing data to JSP pages. JSP pages may handle most of the front-end presentation logic. Business logic may be implemented using enterprise java beans Generally, stateless session beans are used.

Please replace paragraph 85 starting on page 24, with the following rewritten paragraph:

[0085] The kernel will now be described in detail. The kernel enables various services to discover each other on a network and provides a way for services to interact in a dynamic, robust way. No user intervention is required when services are brought on or off line. Services that join the kernel can adapt dynamically when any of the other services go on or off line. Consumers of

the services do not need prior knowledge of the service's implementation. Referring to Fig. 12 , two instances of the kernel, 2530 and 2530', are shown. The architecture of the kernel 2530 features a core 2531, a lease administrator 2532, an RFS server 2533, a codebase server 2534, a registration database 2535, an availability server 2536, and a configuration server 2537. The architecture of the kernel 2530' features a core 2531', a lease administrator 2532', an RFS server 2533', a codebase server 2534', a registration database 2535', the availability server 2536, and the configuration server 2537. Two instances of the kernel are preferably running on separate servers for enhanced availability.